AD-A069 496 NAVAL SURFACE WEAPONS CENTER DAHLGREN LAB VA F/G 9/2
ERROR HANDLING IN THE TRIDENT COMPILER.(U)
SEP 78 H G HUBER
UNCLASSIFIED NSWC/DL-TR-3838 NL

| OF |
AD
A069496

END
DATE
FILMED
7-79
DDC

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ✓ TR-3838	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 ERROR HANDLING IN THE TRIDENT COMPILER.		5. TYPE OF REPORT & PERIOD COVERED 9 FINAL rept. ~~6. PERFORMING ORG. REPORT NUMBER~~
7. AUTHOR(s) 10 Hartmut G./Huber		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Surface Weapons Center ✓ (Code K54) 391 598 Dahlgren, VA 22448		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NIF
11. CONTROLLING OFFICE NAME AND ADDRESS 14 NSWC/DL-TR-3838		11 ~~12. REPORT DATE~~ ~~September 1978~~ ~~13. NUMBER OF PAGES~~ 48
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) 12 40p.		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

context parser
LALR parser
grammar

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper describes briefly the error recovery strategy used within the LALR parser of the TRIDENT Compiler. The main part deals with an analysis of error situations that can occur in a Context Parser. This analysis is based on a classification of terminals, non-terminals, and heads according to certain syntactic characteristics. This information can be encoded in a table which allows error recovery to be table driven. This approach is used in the TRIDENT Compiler for parsing statements and expressions.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

391 598

FOREWORD

During the period of 1975 to 1976, a group within the Fire Control Programming Branch of the FBM Geoballistics Division designed and built a compiler for the TRIDENT Higher Level Language to be used for all fire control programming of the TRIDENT System. The error recovery system described in this report was developed as part of that project. This work was accomplished under task number 36401 and was administratively reviewed by R. M. Pollock, Head of the Programming Branch.

Released by:

[illegible]

RALPH A. NIEMANN, Head
Strategic Systems Department

TABLE OF CONTENTS

1. INTRODUCTION

Any program that processes unscreened input must expect the worst about its input data. If the input data are as complex and, fortunately, as well structured as the source language for a compiler, then a very methodical approach must be used, the most reliable one being parsing according to a grammar.

Compilers use parsers for two purposes. First, to check the source program for syntactic correctness, and second, to control the sequence of semantic actions effecting the translation of the source code into intermediate or object code. This is normally done on several levels: on the lexical level to group characters into tokens, on the level of declarations that inform the compiler about symbols and do not generate code (declaration level), and on the level of constructs that generate code (expression and statement level).

In the TRIDENT Compiler, these three levels are handled by a scanner, an LALR parser, and a context parser (see Section 5). Errors in the source text can occur on any of these levels. Of course, errors can also occur during the execution of semantic routines since by far not all aspects of a programming language are handled syntactically. For example, all checking of compatible attributes of symbols such as OWN, TYPE, PROCEDURE, ARRAY, etc., is implemented by an extra mechanism outside the LALR parser. Similarly, all type checking of expressions is done outside the context parser.

In general, error handling in semantic routines, or parser support routines such as get-next-item, is much simpler than error handling within a parser since those routines deal with a very narrow environment known by the programmer. Therefore, a general method for error handling is to make the grammar used by the compiler more permissive than the true grammar for the language. For example, the compiler grammar will contain productions that allow different ordering of keywords in declarations such as:

```
OWN INTEGER variable list, or

INTEGER OWN variable list
```

or it may allow for a subscripted variable any number of subscripts instead of only one, two, or three. The relevant semantic routines will contain checks for the restrictions according to the true grammar and issue error messages accordingly. This approach is used in the TRIDENT Compiler extensively on the declaration level, to a lesser extent on the expression and statement level.

A parser, on the other hand, operates in a much more general environment and it takes much analysis and work on the programmer's part to classify error situations, to issue meaningful error messages, to take corrective actions, and to recover from the error situation; that is, to find a point to continue parsing. The primary goal of error processing is to inform the user preceisely about the causes of errors in the program and to continue parsing the rest of the program after errors have been found in order to find possibly other independent errors without cascading errors found so far. The purpose is not to second-guess the real intentions of the user and to produce an object program according to what the user probably meant. In accordance with this philosophy, the TRIDENT Compiler will not generate any code once a syntax error has been found on the expression and statement level.

The following basic definitions will be used: An initial substring of the source text is said to contain a syntax error if it cannot be completed to a sentence in the language. The last item of the smallest initial substring with a syntax error is said to be the error item and its place in the source text is the error location.

This paper will explain very briefly the overall error recovery strategy within the LALR parser. The method is very simple and fast and works satisfactorily for a grammer that has almost no nested constructs. For grammars containing mostly nested constructs, a more sophisticated approach is appropriate. A promising approach would be to work out a practical method based on the theoretical scheme for error recovery in conjunction with LR parsers as presented in [1].

The major part of this paper deals with a general method of classifying error situations and corrective actions during error recovery in a context parser. This method is based on a classification of terminals, non-terminals, and heads of an operator grammer, according to certain syntactic characteristics. The scheme is explained in detail for the TRIDENT Higher Level Language (THLL). It allows condensation [1,2] of input text to higher syntactic units if the error item starts a new construct. However, no condensation of partially parsed input on the parser stack is permitted. The reason for this strict condensation policy is to preserve the integrity of the semantic environment. It is this problem of aligning semantic and syntactic processing across syntax errors that has found very little attention in the literature. Yet, in many compilers, including the TRIDENT Compiler, it is important to continue semantic processing even after syntax errors have been encountered since large classes of errors, e.g., type errors, would remain undetected.

This error recovery scheme does not employ backtracking of parsed input and does not pursue parallel parses. It is based on a systematic analysis of grammar units that allows the major error recovery decisions to be made ahead of time for all incorrect programs. It also the production of an error recovery table containing encodings of all important classes of grammar units automatically from a grammar and certain additional information. This aspect and the approach to preserve the integrity of the semantic environment appear to be improvements of existing techniques for syntax error recovery.

2. ERROR RECOVERY IN CONJUNCTION WITH A LALR PARSER

A LALR parser can discover an error item as soon as this item is seen. However, it may be seen when the parser is in a lookahead state. In this case, recognition of the error item is delayed since the LALR parsing tables are optimized in the sense that in lookahead states only those items are considered individually that discriminate between next states, while all other items, legal or not, correspond to a 0 entry in the lookahead list and are treated alike by the lookahead state causing a transition to the same next state. Thus, instead of recognizing the error item at this time, the parser will, based on the assumption that the next item was a legal item not listed in the lookahead list, eventually enter a read state, possibly after a number of apply states, and then recognize the error condition. Thus, an error condition is encountered if the parser, being in a read state S, cannot read the next item.

The general approach in the TRIDENT Compiler is to supply the parser with an item it can read. This is achieved by inserting an item from the read list in the grammar tables into the source text. There is a special algorithm based on the concept of hard tokens (BEGIN, semicolon, END) and priority of tokens that will select a token in the read list. This algorithm has been tuned by trial and error through many experiments. It is most important for a decent behavior of this error recovery scheme. This algorithm may also decide to not select an item. In that case, the error recovery routine will either discard the error item or pop off one element of the parser stack making it the current state. This latter choice is the first corrective action attempt taken by the simple error recovery routine of a standard LALR parser [3].

When a particular error situation causes cascading problems, then it may be possible to make local modifications to the grammar and add error checks to relevant semantic routines. In this way, a small error recovery routine can be tuned up to a powerful debugging aid.

3. OPERATOR GRAMMARS AND CONTEXT PARSERS

The TRIDENT Compiler uses on the statement and expression level a parser which is an improved variation of the classic transition matrix parser as described by D. Gries [4]. This section and the next describe briefly the general behavior of this type of parser, called context parser, both from a syntactic and semantic viewpoint.

Let $G = (V_N, V_T, P, Q)$ be an operator grammar and $Q \rightarrow$ ALPHA Z FINIS the only production in P that involves Q. The terminals ALPHA, FINIS serve as a begin and end marker of a sentence in L(G). We shall consider strings over $V_T \cup V_{IN}$, where $V_{IN} \subset V_N$ is a set of input non-terminals. In praxis, V_{IN} will contain items such as variable-id, procedure-id, etc. Theoretically, V_{IN} can be any subset of V_N and, therefore, the parser described below will work for sentential forms. Thus,

$$L(G) = \{\text{ALPHA S FINIS: } S \in (V_T \cup V_{IN})^* \text{ and } Z \overset{*}{\Rightarrow} S\}$$

Since G is an operator grammar, non-terminals will never be adjacent, neither in rightsides of a production nor in any $S \in L$. We view S as a sequence of items alternating between terminals and non-terminals by filling in a NIL non-terminal, used only for this purpose, between two terminals if they are adjacent.

Consider a parser for L(G) that works as follows: A sentence ALPHA S FINIS in L(G) is parsed by moving through a finite sequence of configurations of the form

$$h_0\ h_1 \ldots h_k,\ V,\ T\ y\ \text{FINIS}$$

where

$h_0 \ldots h_k$ is a sequence of heads,

V is a non-terminal,

T is a terminal, and

y is a string, the remainder string not yet seen by the parser.

We call V the current non-terminal or the item being looked at, $h_0 \ldots h_k$ its left context, and T y FINIS its right context. Each head is an incomplete right part of a production ending in a terminal. A head will be represented in this paper by an initial segment of a right part underlined and marked off, e.g.:

$\underline{\text{variable =}}\lrcorner$ or $\underline{\text{FOR variable = e STEP}}\lrcorner$

The initial configuration is

h_0, NIL, S FINIS

where

h_0 is $\underline{\text{ALPHA}}\rfloor$.

The final configuration is

h_0, Z, FINIS

The parser is initialized such that on input w the initial configuration is

h_0, NIL, w

Thus, a string w parsed successfully as a sentence will have the form:

S FINIS

where

$Z \overset{*}{\Rightarrow} S$

The following moves or transitions from a configuration

---h_{k-1} h_k, V, T V_1T_1---

are possible:

(1) ----h_{k-1} X, V_1, T_1---- X = $\underline{h_kVT}\rfloor$

(2) ------h_{k-1}, X, T_1------ X = h_kVT , V_1 must be NIL

(3) ------h_{k-1}, X, T V_1T_1------ X = h_kV

(4) ---h_{k-1} h_k, X, T_1--- X = VT , V_1 must be NIL

(5) ------h_k X, V_1, T_1------ X = $\underline{VT}\rfloor$

(6) Exit parser if current configuration is h_0, Z, FINIS.

(7) Error condition; attempt to find a resume configuration and continue.

No transitions according to a production $U \rightarrow V$ are ever made. It is implied that the same transition with U as non-terminal is to be used when V is in the non-terminal position. This assumption is made to reduce the size of the parsing tables. Grammars must satisfy certain conditions to allow this parsing scheme. This is explained in more detail in [5].

It is clear that for a given grammar, the number of all possible configurations is, in general, not finite. Some finite procedure is needed to compute the next configuration from the current one. For programming languages, normally a very limited context determines the transition uniquely. For the grammar of the TRIDENT Higher Level Language (THLL), one head to the left and one terminal to the right of the current non-terminal is sufficient. The THLL parser uses a set of computer-generated tables to determine for each triple h, V, T the next configuration. The transition information associated with each triple consists of:

(1) NSYN = syntactic transition number ($1 \leq \text{NSYN} \leq 6$),

(2) NSEM = semantic action number,

(3) NEWX = new head or new non-terminal produced by this transition,

and is called the transition vector for h, V, T. This is described in detail in [5].

For our purpose here, it is only necessary to assume that the parser runs through a sequence of configurations described above and that the next configuration is computed from the configuration base, consisting of the current non-terminal V and some limited left and right context. Such a parser is called a context parser in contrast to an LR(k) parser. We call the sequence of heads to the left of the current non-terminal the head stack. It represents all incomplete rightsides of productions that wait for completion in the reverse order in which they were created.

To simplify the discussion, we assume that the configuration base is a triple. However, the approach is valid in general.

4. SYNTAX DIRECTED SEMANTIC PROCESSING

The main purpose of parsing is to determine syntactic correctness of the input string and to guide semantic processing to effect translation of the input to a semantically equivalent output.

To achieve the syntax directed semantic processing, a semantic routine is associated with each parsing cycle. On the grammar level, this means that a semantic action number NSEM is associated with each head and with each complete rightside. The semantic action SEMANTICS(NSEM) is executed before the corresponding head becomes the new head during the present reduction cycle or before the corresponding rightside is reduced to the leftside non-terminal.

We define a grammar with semantic action numbers as follows:

let $V \rightarrow V_1T_1V_2T_2\ldots V_kT_kV_{k+1}$

be any production (some of the V_i's may be missing) and let $h_1, h_2, \ldots, h_k$ be the heads corresponding to the rightside, from left to right. Then with this production a sequence of numbers $n_1, n_2, \ldots, n_k$, n is associated with the following significance:

n_i, $1 \leq i \leq k$, specifies the semantic action to be taken when h_i is the new head of a syntactic reduction (cases 1 and 5).

n specifies the semantic action when V is the new non-terminal of a syntactic reduction (cases 2, 3, and 4).

It is well known that semantic processing is a major source of difficulty in error recovery. Not only is it necessary after a syntax error to search for a new sound syntactic situation from which to continue, but it is equally important to establish a corresponding semantic situation that makes sense as a basis for continuing semantic processing.

It is assumed here that the semantic situation is completely described by a small set of variables called semantic environment. For the TRIDENT Compiler, this set of variables consists of:

RTOP = a pointer to the top of the stack of operators,

DTOP = a pointer to the top of the stack of operands,

ICFL = a pointer to the last entry into the translated code,

NT = current non-terminal item, and

T = current terminal item.

The first three pointers must be reset to previous values, thus effectively undoing semantics that have begun but could not be completed properly. The other variables can be set to standard values.

Thus, in order to align semantic and syntactic processing across syntactic errors, the semantic environment is saved in certain syntactic situations and is retrieved when this syntactic situation becomes a resume point after processing a syntax error. In the TRIDENT Compiler, the three pointers RTOP, DTOP, and ICFL of the semantic environment are saved uniformly whenever a new head is put on the head stack (NSYN = 1, and NSYN = 5). Thus, the head stack contains with each head the semantic environment that must be reestablished when the corresponding head becomes the top of the head stack after a bracketed construct has effectively been removed and replaced by a default value as a result of a syntax error.

5. SYNTAX ERRORS IN A CONTEXT PARSER

A context parser determines that the present configuration is an error configuration if it cannot compute, according to its tables, a successor configuration. An error configuration

h_0--------H, V, T y

can occur because

A. T is an error item in the source text, or

B. V is an error item in the source text, or

C. V represents a substring in the source text which contains an error item, or

D. H represents a substring in the source text which contains an error item.

For cases C and D, the error item occurred earlier but was not recognized by the parser at that time. This is to be expected with a parser using only limited context to determine the transition, enough for syntactically correct source programs but possibly insufficient for syntactically incorrect ones. However, an error item will always eventually cause the occurrence of an ungrammatical configuration base.

In the following examples corresponding to the four cases listed above, the error item is encircled and the error configuration is indicated:

A. ----BEGIN X = 2 $\textcircled{\text{IF}}$ ----

B. ----BEGIN X = $\textcircled{\text{L}}$: y = 2;----

C. BEGIN INTEGER I; $\textcircled{\text{X}}$ = 3 END FINIS

D. BEGIN INTEGER I; $\textcircled{\text{FOR}}$ X = 1 STEP REAL X END FINIS

Error configurations:

A. -----------------$\underline{\text{BEGIN}}\ \underbrace{\text{var =}}$,   num, IF-----------

B. -----------------$\underline{\text{BEGIN}}\ \underbrace{\text{var =}}$, label-id, :------------

C. ------------$\underbrace{h_0}\ \underbrace{\text{prog-hd SEMIC}}$, assign-e, END FINIS -----

$\underbrace{h_0}$, block-hd, END FINIS

D. $\underbrace{h_0}\ \underbrace{\text{prog-hd SEMIC}}\ \underbrace{\text{FOR...STEP}}$,   D-DCL, END FINIS

The fact that an error situation may not be recognized when the error item is read but possibly much later does not present a problem in the TRIDENT Compiler. Using triples for computing transitions, this case is rare anyway. In fact, it may be advantageous to delay recognition of an error until the next construct starting with a left bracket such as FOR, IF, or (has been parsed. Pinpointing the error item to the programmer may not be quite as straightforward. An error message may for example say:

```
SYNTAX ERROR AT ITEM PRECEDING MOST RECENT IF
EXPRESSION ENDING IN LINE ---.
```

In the TRIDENT Compiler, this technique is needed only for the case where statements and expressions are found in the outermost block where they cannot occur legally.

6. ERROR RECOVERY IN CONJUNCTION WITH THE CONTEXT PARSER OF THE TRIDENT COMPILER

This section describes in some detail the method used to recover from syntax errors in preset statements, executable statements and expressions in the TRIDENT Compiler [6,7]. Appendix A contains the relevant grammar, Appendix B lists terminals together with their names, and Appendix C contains the Error Recovery Table (HTMAP) representing various classes of terminals, non-terminals, and heads.

When the parser recognizes a syntax error, the procedure ERRP is entered. One of the following conditions will exist:

A. The pair H, T is ungrammatical, or

B. The pair H, T is grammatical but the triple H, V, T is not, or

C. The triple H, V, T is grammatical but the transition to the next configuration uses a "marked" head to form a new head or to form a new non-terminal. This means that a substring was being parsed while the parser was in an error condition. This substring may be represented by V or it may already have been discarded at this point, in which case V is an input non-terminal. Thus, ERRP is capable of initiating the parsing of a substring and to take corrective action after that substring is parsed completely. The corrective action consists of reestablishing the semantic environment as it existed when the current head was created. This semantic environment is found on the head stack together with the corresponding head. The non-terminal is left unchanged if it is an input non-terminal; otherwise, it is replaced by:

(1) D-DCL, or

(2) number, or

(3) NIL

depending on which one fits. They are tried in this order. If none fits, then NIL is chosen and regular error processing takes over.

In general, error processing is done for classes of terminals, of non-terminals, and of heads. These classes are defined according to the syntactic characteristics of their elements as follows.

6.1 CLASSES OF TERMINALS

The following classes TC(i), i = 1,..., 7, represent a partition of all terminals. Terminals in the same class are treated alike.

TC(1) = {FINIS, END, SEMIC,), IFEND, CASEEND}

TC(2) = {LOC, BITNOT, NOT}

TC(3) = {GOTO, RETURN, EXIT, LOOPEXIT, PROCEDURE, FOR, SWITCH, PRESET, CASE, IF}

TC(4) = {BEGIN}

TC(5) = {(}

TC(6) = {LB}

TC(7) = {all remaining terminals}

Elements of TC(1) are "right brackets" which initiate a search on the headstack for a corresponding left bracket. When a left bracket is found that matches, then the complete bracketed construct is effectively removed from the configuration. This means, for example, that for the pair ($\underline{\text{BEGIN}}\rfloor$, SEMIC) the last statement is removed, for the pair ($\underline{\text{BEGIN}}\rfloor$, END) the entire block is removed. If the pair (head, terminal) does not match then, except when the head is a begin head, this is interpreted as a complete construct with its right bracket missing. The terminal in TC(1) that caused the backup is not removed in this case. However, if the head is a begin head, then the incomplete statement including the current terminal is removed.

The set TC(1) is partitioned again, each element being in its own class except END and SEMIC being in one class together.

TBR(0) = {FINIS}

TBR(1) = {END, SEMIC}

TBR(2) = {)}

TBR(3) = {IFEND}

TBR(4) = {CASEEND}

Each class TBR(i) of terminal brackets corresponds to a class HDBR(i) of matching head brackets as defined in Section 6.2.

Elements of TC(2) start a new expression that can occur in runtime or compile time (PRESET) statements. Elements of TC(3) start a new statement, or possibly an expression, that is not allowed in PRESETS. For elements of TC(3), the insertion of a semicolon is attempted first. If that does not produce a legal configuration then the current terminal will be given a default head that allows parsing to continue, and the current head on top of the head stack is marked. The technique of using a default head that fits to the current terminal is used for terminals in most classes after certain tests have been made. The default head is (⌋ for terminals in TC(2), WHILE⌋ for LB, and BEGIN⌋ otherwise.

BEGIN cannot be put into the class TC(3) since it may open a block or continue a preset statement already started by the terminal PRESET. Thus, BEGIN does not always start a new construct.

For the left parenthesis two cases must be distinguished. First, it may start a new construct in which case the non-terminal preceding it in the current configuration must be NIL. Second, it may start, together with and depending on the non-terminal preceding it, a subscripted variable, component variable, switch expression, or procedure call. In both cases the current head on top of the head stack is marked and parsing will continue with BEGIN⌋ as the default head for the first case and with (⌋ for the second case.

Elements of TC(7) are discarded; that is, effectively removed from the configuration and the next terminal in the remainder string is made the current one. If this new terminal was not preceded by a non-terminal, other than NIL, then the old non-terminal is kept; otherwise, it is replaced by the new one. This new configuration is checked for legality, or if it can be made legal by replacing the current non-terminal by D-DCL, or NUMBER, or NIL. If not, then it is treated according to the described classification of the current terminal.

6.2 CLASSES OF HEADS

Each of the classes defined below represents the collection of those heads that have absorbed a specific left bracket as indicated in the following table.

Head Class	Left Bracket Absorbed	Corresponding Right Bracket
HDBR(0)	-	FINIS
HDBR(1)	BEGIN	END ;
HDBR(2)	(	)

Head Class	Left Bracket Absorbed	Corresponding Right Bracket
HDBR(3)	IF	IFEND
HDBR(4)	CASE	CASEEND
HDBR(5)	LB	RB same as)

HDC(0) = {h_0}

HDC(1) = {PROG-HD SEMIC, BLOCK-HD SEMIC, BEGIN,
PRESET-HD SEMIC, PRESET BEGIN}

HDC(2) = {LP, ARRAY-ID LP, STACK-ID LP, PROC-ID LP,
COMP-ID LP, GOTO SWITCH-ID LP, VAR-HD COMMA,
COMP-HD COMMA, FUN-EXP-HD COMMA}

HDC(3) = {IF, IF E THEN, IF-HD COMMA, IF-HD COMMA E THEN,
IF-HD ELSE}

HDC(4) = {CASE, CASE E DO, CASE-HD COMMA, }

HDC(5) = {WHILE E LB, FOR VAR-ID = E STEP E UNTIL E LB,
FOR VAR-ID = E STEP E WHILE E LB,
FOR VAR-ID = E REPEAT E WHILE E LB,
LOOP-HD1 COMMA, LOOP-HD2 COMMA,
LOOP-HD3 COMMA, LOOP-HD4 COMMA}

All elements of one head class will match one specific right bracket, in one case two right brackets (END ;). Conversely, to each right bracket corresponds one unique head class with the exception of the right parenthesis. The terminals) and } have not been distinguished syntactically; hence, the conflict concerning their head classes must be resolved by a special test. A situation like this should be avoided in the design of a grammar.

6.3 CLASSES OF NON-TERMINALS

For language constructs that have a repetitive substructure such as IF expressions, CASE expressions, or blocks, the grammar is designed such that a non-terminal represents an initial string to which, recursively, a new substructure can be added on, or that can be closed by a "right bracket" terminal. For example:

CASE-HD → CASE E DO E-ST

CASE-HD → CASE-HD COMMA E-ST

CASE-E → CASE-HD CASEEND

Therefore, non-terminals of this kind must not be discarded without having been closed by a right bracket. The classes NTC(1) and NTC(2) are of this type. A closer look at the grammar shows that such a non-terminal U is always formed by a transition using U → H V from a configuration

---h'H, V, T y

where H, V, T was a grammatical triple. Otherwise, this transition could not have been made. Therefore, the resulting configuration

h', U, T y

is in error only because of h'. The proper treatment, then, is to supply a default head for U, T and to complete the current construct whose initial segment is represented by U. After that, the error in h' will come up again. The default head for elements of NTC(1) is BEGIN, and for elements of NTC(2) is FUN-EXP-HD COMMA.

NTC(1) = {PROG-HD, BLOCK-HD, PRESET-HD, IF-HD, CASE-HD, VAR-HD, COMP-HD, FUN-EXP-HD}

NTC(2) = {LOOP-HD1, LOOP-HD2, LOOP-HD3, LOOP-HD4}

Finally, the following class

NTLP(1) = {ARRAY-ID, STACK-ID, PROC-ID, SWITCH-ID, COMP-ID}

contains those non-terminals that can precede a left parenthesis and must be associated with this left parenthesis to start a new construct. No other non-terminal apart from the filler NIL can precede a left parenthesis.

6.4 SINGULAR CASE

All non-terminals in NTC(1) and NTC(2) have absorbed a left bracket. The case can occur that a non-terminal absorbs a right bracket. This happens in the THLL grammar only once:

LABEL-END → LABEL-ID : END

When an error occurs with LABEL-END in the non-terminal position, then the non-terminal is replaced by NIL and an END is inserted. This corresponds to replacing a labeled END by an unlabeled END.

6.5 IMPLEMENTATION CONSIDERATIONS

The classes of terminals, non-terminals, and heads defined above can be implemented economically and accessed very efficiently using a single one-dimensional array HTMAP containing N words where N = max (number of heads, number of non-terminals, number of terminals). Each class name C corresponds to a field in a word. If I is the numeric code for a terminal, or a head, then

$$C[\text{LOC HTMAP (I)}] = \begin{cases} \text{J if I belongs to C(J)} \\ \text{0 otherwise.} \end{cases}$$

In the same manner, default heads associated with terminals can be encoded in this array. Alternatively, default heads associated with a class can be encoded directly in the field for that class. This is done in the TRIDENT Compiler for NTC. The field for NTC contains the encoding of $\underline{\text{BEGIN}}\rfloor$ for all elements of NTC(1); it contains the encoding of $\underline{\text{FUN-EXP-HD COMMA}}\rfloor$ for all elements of NTC(2). In the TRIDENT Compiler, HTMAP is an array of 74 32-bit words. This array also contains other information indicated by SCHEMA. It is used for generating a file that is needed by a Symbolic Debug System TOADC [8].

Besides such a classification device for terminals, non-terminals, and heads, a set of basic utility procedures is needed for at least the following purposes.

A. Checking if a given pair (head, terminal) or a given triple (head, non-terminal, terminal) is legal. Computing the transition vector for a given triple.

B. To allow the replacement of the current non-terminal by a filling non-terminal that makes the resulting triple legal.

C. To provide a general capability of inserting items into the sequence of items as seen by the parser.

D. Debugging aids to allow, depending on options, the printing of parsing cycles (configuration base and transition vector) of the entire head stack, the entire D-stack, the entire R-stack, and selected portions of the translated code.

All these routines and some others, more special ones, are employed in the syntactic error recovery procedure of the TRIDENT Compiler.

7. CONCLUSION

This paper describes some principles on which the recovery from syntactic errors encountered in a context parser are based. Both parsing and error recovery are table driven. This provides a unified and reliable approach for handling a wide variety of problems that can be formulated as translation of a language described by an operator grammar.

In the TRIDENT compiler, the table HTMAP used in error recovery was constructed by hand, while the parsing tables were generated by a general program from a grammar. From the definition of the various classes defined in the previous sections, it can be seen that these classes can be formed and encoded into an error recovery table automatically from the grammar and a high level description of some characteristics concerning the meaning of grammar symbols. It is planned to incorporate the construction of an error recovery table into the LISP program that currently generates the parsing tables.

REFERENCES

1. M. Dennis Mickunas and John A. Modry, *Automatic Error Recovery for LR Parsers*, CACM, Vol. 21, No. 6, June 1978.

2. Susan L. Graham and Steven P. Rhodes, *Practical Syntactic Error Recovery*, CACM, Vol. 18, No. 11, November 1975.

3. J.E. Musinski, *Lookahead Recall Error Recovery for LALR Parsers*, SIGPLAN Notices, October 1977.

4. D. Gries, *Compiler Construction for Digital Computers*, John Wiley and Sons, Inc., 1971.

5. H. G. Huber, *Generating Parsing Tables for Context Parsers*, Forthcoming.

6. The THLL Group, *TRIDENT Higher Level Language User's Guide*, Technical Report 3657, July 1977, Revised May 1978.

7. Paul Shebalin, *TRIDENT Higher Level Language Syntax Definition*, NSWC/DL TN-K-9/78.

8. J. A. Gaines, Jr., *An External Debugging System for Weapons System Programs Written in a Higher Level Language*, Proceedings, COMPSAC 78.

APPENDIX A

TRIDENT STATEMENT AND EXPRESSION GRAMMAR

TRIDENT STATEMENT AND EXPRESSION GRAMMAR

The grammar as listed in this appendix represents the input data for a LISP program to produce parsing tables. It consists of the following six groups of data:

(1) List of options

(2) List of terminals; they will be assigned a numerical code, in the order in which they appear

(3) Goal, highest syntactic unit

(4) List of productions together with semantic action numbers

(5) List of defined terminals

(6) List of input non-terminals

As usual in LISP, a list of items is written as

```
(X1  X2  X3...)
```

where each item X_i can be a symbol or a number or a list. A production

$Y \rightarrow X1\ \ X2 \ldots Xn$

is represented as

```
(Y  X1  X2...Xn)
```

Each production is followed by the list of the associated semantic action numbers. The action number 10 specifies a no action.

```
(OFSYSF CPFILE T T T T T T)

(ALPHA ADD-OP MULT-OP ** BITOR-XOR BITAND BITNOT LOC ENTRYP REL-OP
NOT AND OR XOR = LP RP WHILE FOR STEP UNTIL REPEAT DO CASE CASEEND
COMMA IF THEN ELSE  IFEND COLON SEMIC GOTO RETURN EXIT LOOPEXIT
PROCEDURE BEGIN END NULL SWITCH PRESET TO FINIS
PRES-=    LB)

(

(
(S          ALPHA      PROG       FINIS                                      )
            (10   70)
(PROG       PROG-HD    END                                                   )
            (69)
(PROG       PROG-HD    SEMIC      END                                        )
            (10   69)
(PROG-HD    BEGIN      DCL                                                   )
            (65   10)
(PROG-HD    PROG-HD    SEMIC      DCL                                        )
            (10   10)

(DCL        DATA-DCL                                                         )
            (10)
(DCL        PROC-DCL                                                         )
            (10)


(DATA-DCL D-DCL                                                              )
            (10)
(DATA-DCL SW-DCL                                                             )
            (10)
(DATA-DCL PRESET-DCL                                                         )
            (10)

(SW-DCL     SWITCH     SWITCH-ID  =          LAB-LIST                        )
            (59   10   60)
(LAB-LIST LABEL-ID                                                           )
            (10)
(LAB-LIST LAB-LIST   COMMA      LABEL-ID                                     )
            (51   10)


(PRESET-DCL PRESET   PRESET-EL                                               )
            (71   72)
(PRESET-DCL PRESET-HD  END                                                   )
            (72)
(PRESET-DCL PRESET-HD  SEMIC  END                                            )
            (10   72)
(PRESET-HD  PRESET   BEGIN      PRESET-EL                                    )
            (71   10   10)
(PRESET-HD  PRESET-HD  SEMIC    PRESET-EL                                    )
            (10   10)

(PRESET-EL  VAR        PRES-=     S-E-LIST                                   )
            (74   75)
(PRESET-EL  VAR        TO         VAR        PRES-=     S-E-LIST             )
            (76   74   75)
(S-E-LIST SIMPLE-E                                                           )
            (10)
(S-E-LIST S-E-LIST   COMMA      SIMPLE-E                                     )
            (73   10)
```

THIS PA[illegible] QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

```
(PROC-DCL PROCEDURE PROC-ID    SEMIC       PROC-BODY                     )
          (61   62    63)
(PROC-BODY E                                                             )
          (10)
(PROC-BODY PROPER-ST                                                     )
          (10)
(PROC-BODY RETURN-ST                                                     )
          (10)


(E        SIMPLE-E                                                       )
          (10)
(E        ASSIGN-E                                                       )
          (10)

(ST       CC-ST                                                          )
          (10)
(ST       PROPER-ST                                                      )
          (10)
(CC-ST    GOTO-ST                                                        )
          (10)
(CC-ST    EXIT-ST                                                        )
          (10)
(CC-ST    RETURN-ST                                                      )
          (10)

(GOTO-ST  GOTO       LABEL-ID                                            )
          (17    16)
(GOTO-ST  GOTO       SWITCH-ID LP         E          RP                  )
          (17    10    19)

(EXIT-ST  EXIT                                                           )
          (10    45)
(EXIT-ST  EXIT       LABEL-ID                                            )
          (10    45)
(EXIT-ST  LOOPEXIT                                                       )
          (10    46)
(EXIT-ST  LOOPEXIT   LABEL-ID                                            )
          (10    46)
(RETURN-ST RETURN                                                        )
          (10    47)
(RETURN-ST RETURN    E                                                   )
          (10    47)


(PROPER-ST BLOCK-ST                                                      )
          (10)
(PROPER-ST LOOP-ST                                                       )
          (10)
(PROPER-ST NULL-ST                                                       )
          (10)

(SIMPLE-E BOOL-E                                                         )
          (10)
```

```
(BOOL-E     BOOL-TRM                                                   )
            (10)
(BOOL-TRM   BOOL-FAC                                                   )
            (10)      ..
(BOOL-TRM   BOOL-TRM   OR          BOOL-FAC                            )
            (11   12)
(BOOL-TRM   BOOL-TRM   XOR         BOOL-FAC                            )
            (1    15)
(BOOL-FAC   BOOL-SEC                                                   )
            (10)
(BOOL-FAC   BOOL-FAC   AND         BOOL-SEC                            )
            (13   14)
(BOOL-SEC   BOOL-PRI                                                   )
            (10)
(BOOL-SEC   NOT         BOOL-PRI                                       )
            (1    16)
(BOOL-PRI   RELAT                                                      )
            (10)
(BOOL-PRI   S-ARTH-E                                                   )
            (10)
(RELAT      S-ARTH-E   REL-OP      S-ARTH-E                            )
            (1    2)

(S-ARTH-E   TERM                                                       )
            (10)
(S-ARTH-E   ADD-OP    TERM                                             )
            (1    6)
(S-ARTH-E   S-ARTH-E   ADD-OP      TERM                                )
            (1    2)
(TERM       FACTOR                                                     )
            (10)
(TERM       TERM       MULT-OP     FACTOR                              )
            (1    2)
(FACTOR     PRIM4                                                      )
            (10)
(FACTOR     FACTOR     **          PRIM4                               )
            (1    7)
(PRIM4      PRIM3                                                      )
            (10)
(PRIM4      PRIM4      BITOR-XOR PRIM3                                 )
            (1    2)
(PRIM3      PRIM2                                                      )
            (10)
(PRIM3      PRIM3      BITAND      PRIM2                               )
            (1    2)
(PRIM2      PRIMARY                                                    )
            (10)
(PRIM2      BITNOT     PRIMARY                                         )
            (1    3)

(PRIMARY    NUM                                                        )
            (10)
(PRIMARY    STRING                                                     )
            (10)
(PRIMARY    VAR                                                        )
            (10)
(PRIMARY    LOC        VAR                                             )
            (1    29)
(PRIMARY    LOC        PROC-ID                                         )
            (1    29)
(PRIMARY    LOC        FORMAT-ID                                       )
            (1    29)
(PRIMARY    FUN-EXP                                                    )
            (10)
(PRIMARY    COND-E                                                     )
            (10)
(PRIMARY    CASE-E                                                     )
            (10)
(PRIMARY    LP         E           RP                                  )
            (10   10)
```

```
(ASSIGN-E VAR       =         E                                              ;
          (5   6)


(COND-E   IF-HD     IFEND                                                    ;
          (24)
(COND-E   IF-HD     ELSE      E-ST      [FEND]                               ;
          (23   24)
(IF-HD    IF        E         THEN      E-ST                                 ;
          (20   21   10)
(IF-HD    IF-HD     COMMA     E         THEN      E-ST                       ;
          (22   21   10)


(CASE-E   CASE-HD   CASEEND                                                  ;
          (28)
(CASE-HD  CASE      E         DO        E-ST                                 ;
          (25   26   10)
(CASE-HD  CASE-HD   COMMA     E-ST                                           ;
          (27   10)


(BLOCK-ST BLOCK-HD  END                                                      ;
          (68)
(BLOCK-ST BLOCK-HD  SEMIC     END                                            ;
          (66   67)
(BLOCK-ST BLOCK-HD  SEMIC     LABEL-END                                      ;
          (66   67)
(BLOCK-HD PROG-HD   SEMIC     LABEL-E-ST                                     ;
          (10   10)
(BLOCK-HD BEGIN     LABEL-E-ST                                               ;
          (65   10)
(BLOCK-HD BLOCK-HD  SEMIC     LABEL-E-ST                                     ;
          (66   10)


(LOOP-ST  WHILE     E         DO        E-PST                                ;
          (30   31   32)
(LOOP-ST  FOR  VAR-ID  = E   STEP  E   UNTIL  E   DO  E-PST                  ;
          (33   10   34   35   36   37)   ,
(LOOP-ST  FOR  VAR-ID  = E   STEP  E   WHILE  E   DO  E-PST                  ;
          (33   10   34   38   39   40)
(LOOP-ST  FOR  VAR-ID  = E   REPEAT  E   WHILE  E   DO  E-PST                ;
          (33   10   41   42   43   44)

(NULL-ST  NULL                                                               ;
          (9)
(E-PST     E                                                            )
           (10)
(A-PST     PROPER-ST                                                    )
           (10)
(E-ST      E-PST                                                        )
           (10)
(E-ST      CC-ST                                                        )
           (10)
(LABEL-E-ST E-ST                                                        )
           (10)
(LABEL-E-ST LABEL-ID  COLON  LABEL-E-ST                                 )
           (64    10)

(LABEL-END) LABEL-ID COLON      END                                     )
           (64    10)
(LABEL-END) LABEL-ID COLON      LABEL-END                               )
           (64    10)

(VAR       VAR-ID                                                       )
           (10)
(VAR       STACK-ID  LP         E          2)                           )
           (53    55)
(VAR       A-SUB-VAR                                                    )
           (10)
(VAR       COMP-VAR                                                     )
           (10)
(A-SUB-VAR VAR-HD    RP                                                 )
           (52)
(VAR-HD    ARRAY-ID  LP         E                                       )
           (50    10)
(VAR-HD    VAR-HD    COMMA      E                                       )
           (51    10)
(COMP-VAR   COMP-HD   RP                                                )
           (56)
(COMP-HD   COMP-ID   LP         E                                       )
           (55    10)
(COMP-HD   COMP-HD   COMMA      E                                       )
           (51    10)

(FUN-EXP   PROC-ID                                                      )
           (10)
(FUN-EXP   FUN-EXP-HD  RP                                               )
           (53)
(FUN-EXP-HD  PROC-ID  LP    A-PAR                                       )
           (57    10)
(FUN-EXP-HD  FUN-EXP-HD  COMMA   A-PAR                                  )
           (48    10)
```

```
(A-PAR     E                                                          )
           (10)
(A-PAR     ARRAY-ID                                                   )
           (10)
(A-PAR     STACK-ID                                                   )
           (10)
(A-PAR     DEV-ID                                                     )
           (10)
(A-PAR     ENTRYP     PROC-ID                                         )
           (1    8)
(A-PAR     FORMAT-ID                                                  )
           (10)
(A-PAR     INTER-ID                                                   )
           (10)
(A-PAR     LOOP-ARG                                                   )
           (10)


(LOOP-ARG LOOP-HD1   RP                                               )
           (82)
(LOOP-ARG LOOP-HD2   RP                                               )
           (83)
(LOOP-ARG LOOP-HD3   RP                                               )
           (84)
(LOOP-ARG LOOP-HD4   RP                                               )
           (85)
(LOOP-HD1    WHILE    E           LB          A-PAR                   )
           (30    31    10)
(LOOP-HD2    FOR  VAR-ID  =  E   STEP  E   UNTIL  E   LB  A-PAR       )
           (33    10   34    35    36   10)
(LOOP-HD3    FOR  VAR-ID  =  E   STEP  E   WHILE  E   LB  A-PAR       )
           (33    10   34    38    39   10)
(LOOP-HD4    FOR  VAR-ID  =  E   REPEAT  E   WHILE  E   LB  A-PAR     )
           (33    10   41    42    43   10)
(LOOP-HD1    LOOP-HD1  COMMA  A-PAR                                   )
           (81    10)
(LOOP-HD2    LOOP-HD2  COMMA  A-PAR                                   )
           (81    10)
(LOOP-HD3    LOOP-HD3  COMMA  A-PAR                                   )
           (81    10)
(LOOP-HD4    LOOP-HD4  COMMA  A-PAR                                   )
           (81    10)
                                                                      )

(MULT-OP ADD-OP REL-OP                                                )

(NUM   STRING  VAR-ID  ARRAY-ID  STACK-ID  PROC-ID  DEV-ID
   FORMAT-ID  LABEL-ID  SWITCH-ID . INTER-ID COMP-ID O-DCL            )

   PRINT (ENDPARSERG)
   STOP))))))))))))
   FIN
```

APPENDIX B

NAMES OF TERMINALS

NAMES OF TERMINALS

Names	Terminals
ADD-OP	ADD-OP
MULT-OP	MULT-OP
**	**
BITOR-XOR	BITOR-XOR
BITAND	BITAND
BITNOT	BITNOT
LOC	LOC
ENTRYP	ENTRYP
REL-OP	REL-OP
NOT	NOT
AND	AND
OR	OR
XOR	XOR
=	=
LP	(
RP	) or }
WHILE	WHILE
FOR	FOR
STEP	STEP
UNTIL	UNTIL
REPEAT	REPEAT
DO	DO
CASE	CASE
CASEEND	CASEEND
COMMA	,
IF	IF
THEN	THEN
ELSE	ELSE
IFEND	IFEND
COLON	:
SEMIC	;
GOTO	GOTO
RETURN	RETURN
EXIT	EXIT
LOOPEXIT	LOOPEXIT
PROCEDURE	PROCEDURE
BEGIN	BEGIN
END	END
NULL	NULL
SWITCH	SWITCH
PRESET	PRESET
TO	TO
FINIS	FINIS
PRES-=	=
LB	{

APPENDIX C

ERROR RECOVERY TABLE HTMAP

```
[illegible] GHMAP
[illegible]MAP
   BEGIN
   /*
     NONTERMS     HEADS                                      TERMINALS

      0 NIL         ALPHA
      1 NUM         PROG-NO   SEMIC                          ADD-OP
      2 STRING      BEGIN                                    MULT-OP
      3 VAR-ID      SWITCH                                   **
      4 ARRAY-ID    SWITCH   SWITCH-ID  =                    BITOR-XOR
      5 STACK-ID    LAB-LIST   COMMA                         BITAND
      6 PROC-ID     PRESET                                   BITNOT
      7 CAV-ID      PRESET-NO   SEMIC                        LOC
      8 FORMAT-ID   PRESET  BEGIN                            ENTRYP
      9 LABEL-ID    VAR    PRES-=                            REL-OP
     10 SWITCH-ID   VAR    TO                                NOT
     11 INTGR-ID    VAR    TO   VAR    PRES-=                AND
     12 COMP-ID     S-E-LIST   COMMA                         OR
     13 C-DCL       PROCEDURE                                XOR
     14 G           PROCEDURE  PROC-ID   SEMIC               =
     15 PROG        GOTO                                     LP
     16 PROG-NO     GOTO  SWITCH-ID  LP                      RP
     17 DCL         EXIT                                     WHILE
     18 DATA-DCL    LOOPEXIT                                 FOR
     19 SW-DCL      RETURN                                   STEP
     20 LAB-LIST    BOOL-TRM  OR                             UNTIL
     21 PRESET-DCL  BOOL-TRM  XOR                            REPEAT
     22 PRESET-NO   BOOL-FAC AND                             DO
     23 PRESET-EL   NOT                                      CASE
     24 S-E-LIST    S-ARTH-E  REL-OP                         CASEEND
     25 PROC-DCL    ADD-OP                                   COMMA
     26 PROC-BODY   S-ARTH-E  ADD-OP                         IF
     27 E           TERM   MULT-OP                           THEN
     28 ST          FACTOR   **                              ELSE
     29 CO-ST       PRIMN  BITOR-XOR                         SPEND
     30 GOTO-ST     PRIMS  BITAND                            COLON
     31 EXIT-ST     BITNOT                                   SEMIC
     32 RETURN-ST   LOC                                      GOTO
     33 PROPER-ST   LP                                       RETURN
     34 SIMPLE-E    VAR   =                                  EXIT
     35 BOOL-E      IF-NO  ELSE                              LOOPEXIT
     36 BOOL-TRM    IF                                       PROCEDURE
     37 BOOL-FAC    IF  E   THEN                             BEGIN
     38 BOOL-SEC    IF-NO  COMMA                             END
     39 BOOL-PRI    IF-NO  COMMA  E    THEN                  NULL
     40 RELAT       CASE                                     SWITCH
     41 S-ARITH-E   CASE  E   DO                             PRESET
     42 TERM        CASE-NO  COMMA                           TO
     43 FACTOR      BLOCK-NO  SEMIC                          FINIS
     44 PRIMN       WHILE                                    PRES-=
     45 PRIMS       WHILE  E   DO                            LB
     46 PRIME       FOR
     47 PRIMARY     FOR   VAR-ID  =
     48 ASSIGN-E    FOR   VAR-ID  =  E    STEP
     49 COND-E      FOR   VAR-ID  =  E    STEP  E   UNTIL
     50 IF-NO       FOR   VAR-ID  =  E    STEP  E   UNTIL  E   DO
     51 CASE-E      FOR   VAR-ID  =  E    STEP  E   WHILE
     52 CASE-NO     FOR   VAR-ID  =  E    STEP  E   WHILE  E   DO
     53 BLOCK-ST    FOR   VAR-ID  =  E    REPEAT
     54 BLOCK-NO    FOR   VAR-ID  =  E    REPEAT  E   WHILE
     55 LOOP-ST     FOR   VAR-ID  =  E    REPEAT  E   WHILE  E   DO
     56 NULL-ST     LABEL-ID
     57 E-PST       STACK-ID  LP
     58 E-ST        ARRAY-ID  LP
     59 LABEL-E-ST  VAR-NO  COMMA
     60 LABEL-END   COMP-ID  LP
     61 VAR         COMP-NO  COMMA
     62 A-SUB-VAR   PROC-ID  LP
     63 VAR-NO      FUN-EXP-NO  COMMA
     64 COMP-VAR    ENTRYP
     65 COMP-NO     WHILE  E   LB
     66 FUN-EXP     FOR   VAR-ID  =  E    STEP  E   UNTIL  E   LB
     67 FUN-EXP-NO  FOR   VAR-ID  =  E    STEP  E   WHILE  E   LB
     68 A-PAR       FOR   VAR-ID  =  E    REPEAT  E   WHILE  E   LB
     69 LOOP-ARG    LOOP-NO1  COMMA
     70 LOOP-NO1    LOOP-NO2  COMMA
     71 LOOP-NO2    LOOP-NO3  COMMA
     72 LOOP-NO3    LOOP-NO4  COMMA
     73 LOOP-NO4
   OWN INTEGER ARRAY HTMAP(73);
   OWN POINTER HTORG;
   GLOBAL HTORG;

PRESET BEGIN
   HTORG = LOC HTMAP(0);

/*            K29                   K21   K20  K13   K10   K3    K3*/
/*          SCHEMA                 HDBR  NTLP NTC    TBR   HD    TC*/
HTMAP( 0) =  2K29+                 0K21+                        0 ;
HTMAP( 1) =  2K29+                 1K21+                        7 ;
HTMAP( 2) =  2K29+                 1K21+                        7 ;
HTMAP( 3) =  2K29+                                              7 ;
HTMAP( 4) =  2K29+                       1K20+                  7 ;
HTMAP( 5) =  2K29+                       1K20+                  7 ;
HTMAP( 6) =  2K29+                       1K20+           33K3 + 2 ;
HTMAP( 7) =  2K29+                 1K21+                 33K3 + 2 ;
HTMAP( 8) =  2K29+                 1K21+                        7 ;
HTMAP( 9) =  1K29+                                              7 ;
HTMAP(10) =  2K29+                       1K20+           33K3 + 2 ;
HTMAP(11) =  1K29+                                              7 ;
HTMAP(12) =  1K29+                       1K20+                  7 ;
HTMAP(13) =  2K29+                                              7 ;
HTMAP(14) =  2K29+                                              7 ;
HTMAP(15) =  2K29+                                        2K3 + 5 ;
HTMAP(16) =  1K29+                 2K21+      2K13+ 2K10+       1 ;
HTMAP(17) =  2K29+                                              7 ;
HTMAP(18) =  2K29+                                        2K3 + 3 ;
HTMAP(19) =  1K29+                                              7 ;
HTMAP(20) =  1K29+                                              7 ;
HTMAP(21) =  1K29+                                              7 ;
HTMAP(22) =  1K29+                            2K13+             7 ;
HTMAP(23) =  1K29+                                        2K3 + 3 ;
HTMAP(24) =  1K29+                                  4K10+       1 ;
HTMAP(25) =  1K29+                                              7 ;
HTMAP(26) =  1K29+                                        2K3 + 3 ;
HTMAP(27) =  1K29+                                              7 ;
HTMAP(28) =  1K29+                                              7 ;
HTMAP(29) =  1K29+                                  3<10+       1 ;
HTMAP(30) =  1K29+                                              7 ;
HTMAP(31) =  1K29+                                  1K10+       1 ;
HTMAP(32) =  1K29+                                        2K3 + 3 ;
HTMAP(33) =  0K29+                 2K21+                  2K3 + 3 ;
HTMAP(34) =  1K29+                                        2K3 + 3 ;
HTMAP(35) =  0K29+                 3K21+                  2K3 + 3 ;
HTMAP(36) =  1K29+                 3K21+                  2K3 + 3 ;
HTMAP(37) =  0K29+                 3K21+                  2K3 + 4 ;
HTMAP(38) =  1K29+                 3K21+            1K10+       1 ;
HTMAP(39) =  0K29+                 3K21+                        7 ;
HTMAP(40) =  1K29+                 4K21+                  2K3 + 3 ;
HTMAP(41) =  0K29+                 4K21+                  2K3 + 3 ;
HTMAP(42) =  0K29+                 4K21+                        7 ;
HTMAP(43) =  2K29+                 1K21+            0<10+       1 ;
HTMAP(44) =  1K29+                                              7 ;
HTMAP(45) =  2K29+                                       44K3 + 6 ;
HTMAP(46) =  1K29+                                               0 ;
HTMAP(47) =  1K29+                                               0 ;
HTMAP(48) =  1K29+                                               0 ;
HTMAP(49) =  1K29+                                               0 ;
HTMAP(50) =  2K29+                                 2K13+         0 ;
HTMAP(51) =  1K29+                                               0 ;
HTMAP(52) =  2K29+                                 2K13+         0 ;
HTMAP(53) =  1K29+                                               0 ;
HTMAP(54) =  1K29+                                 2K13+         0 ;
HTMAP(55) =  2K29+                                               0 ;
HTMAP(56) =  2K29+                                               0 ;
HTMAP(57) =  1K29+                     2K21+                     0 ;
HTMAP(58) =  1K29+                     2K21+                     0 ;
HTMAP(59) =  1K29+                     2K21+                     0 ;
HTMAP(60) =  1K29+                     2K21+                     0 ;
HTMAP(61) =  1K29+                     2K21+                     0 ;
HTMAP(62) =  1K29+                     2K21+                     0 ;
HTMAP(63) =  1K29+                     2K21+       2K13+         0 ;
HTMAP(64) =  1K29+                                               0 ;
HTMAP(65) =  1K29+                     5K21+       2K13+         0 ;
HTMAP(66) =  1K29+                     5K21+                     0 ;
HTMAP(67) =  1K29+                     5K21+       2K13+         0 ;
HTMAP(68) =  1K29+                     5K21+                     0 ;
HTMAP(69) =  1K29+                     5K21+                     0 ;
HTMAP(70) =  1K29+                     5K21+      63K13+         0 ;
HTMAP(71) =  1K29+                     5K21+      63K13+         0 ;
HTMAP(72) =  1K29+                     5K21+      63K13+         0 ;
HTMAP(73) =  0K29+                                63K13+         0 ;
END ;

  END
  FINIS
```

DISTRIBUTION

Virginia Polytechnic Institute
and State University
Department of Computer Science
562 McBryde Hall
Blacksburg, VA 24060
ATTN: Dr. R. Nance
Dr. J. Lee

Institute for Defense Analysis
400 Army Navy Drive
Arlington, VA 22202
ATTN: Dr. D. Fisher

Defense Documentation Center
Cameron Station
Alexandria, VA 22314 (12)

Library of Congress
Washington, DC 20540
ATTN: Gift and Exchange Division (4)

K54 (25)
X210 (2)